This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (Currently Amended) A system for dynamically detecting potential race conditions in a program having ~~a plurality of threads~~ at least one thread and one or more shared memory locations, the system comprising:

a processor, wherein the processor is adapted to:

~~with respect to each shared memory location, (i) a mechanism for maintaining a set of concurrent thread segments that access said shared memory location, and (ii) a mechanism for maintaining a first set of locks associated with said shared memory location;~~

~~with respect to each thread, (i) a mechanism for maintaining a set of thread segments that are ordered before a current thread segment of a thread, (ii) a mechanism for maintaining a second set of locks that are acquired and released by the thread, and maintaining a virtual clock associated with each thread; and~~

~~a mechanism for reporting a warning when a potential race condition is detected~~

initialize a virtual clock (C_t) respectively for each of the at least one thread;

initialize a set of candidate locks for each of at least one shared memory location (S_x);

initialize a set of locks (S_t) held for each of at least one thread;

initialize a set of concurrent thread segments (T_x) accessing each of the at least one shared memory location;

initialize an ordered set of thread segments (B_t) below a current thread, wherein the ordering is achieved using the virtual clock associated with each thread;

upon detection of a fork of a new concurrent thread (t1) by a prior thread (t):

increment the virtual clock associated with the prior thread (t);

initialize the virtual clock associated with the new concurrent thread (t1);

update the ordered set of thread segments (B_t) below the prior thread (t) by forming a union of a current ordered set for the prior thread (t) and a singleton set having a thread segment <t, C_t> comprising a tuple of a thread identifier and a virtual clock time;

upon detection of a join call by the prior thread (t):

update the ordered set of thread segments (B_t) by forming a union of the ordered set of thread segments (B_t), a second ordered set of thread segments (B_t1) below thread (t1) that do not belong to the prior thread (t), and a singleton set containing a current thread segment associated with the current thread (t1); and

upon detection of a read or write to a memory location (x), determining whether a potential race condition exists.

2. (Cancelled)

3. (Currently Amended) The system of claim 1 wherein ~~the mechanism for maintaining the set of thread segments that are ordered before the current thread segment comprises a mechanism for maintaining a set of ordered pairs, wherein one member of a pair in the set of ordered pairs is a thread identifier, and the other member of the pair is the virtual clock value associated with the thread identified by the thread identifier~~the processor is further adapted to upon detection of a read or write of a memory location (x) by a thread (t):

update a set of concurrent thread segments concurrently accessing the memory location (T_x) after the read or write;

if a cardinality of the set of concurrent thread segments (T_x) is less than or greater than 1, set a current value of a lockset (S_t) to a current value of the set of candidate locks for the memory location (S_x);

if the cardinality of the set of concurrent thread segments is greater than 1, set a new value of the set of candidate locks for the memory location (S_x) to an intersection of a prior value of the set of candidate locks for the memory location (S_x) and the lockset (S_t);

if the set of candidate locks (S_x) is empty and a new value of cardinality of the set of concurrent thread segments (T_x) is greater than 1, reporting a potential race condition.

4. (Currently Amended) A computer-implemented method for dynamically detecting a potential race condition in a program having a plurality of threads and one or more shared memory locations, the method comprising:

with respect to each shared memory location,

maintaining a first set of locks associated with said each shared memory location, and

maintaining a set of concurrent thread segments that access said each shared memory location;

with respect to each thread,

maintaining a respective virtual clock associated with each thread;

maintaining a second set of locks that are acquired and released by a thread, and

maintaining a set of thread segments that are ordered before a current thread segment of the thread, wherein the thread segments are ordered using each associated respective virtual clock;

upon detection of a fork of a new concurrent thread (t1) by a prior thread (t):

incrementing the virtual clock associated with the prior thread (t);

initializing the virtual clock associated with the new concurrent thread (t1);

updating the ordered set of thread segments (B_t) below the prior thread (t) by forming a union of a current ordered set for the prior thread (t) and a singleton set having a thread segment <t, C_t> comprising a tuple of a thread identifier and a virtual clock time;

upon detection of a read or write to a memory location (x), determining whether a potential race condition exists.

5. (Canceled)

6. (Previously Presented) The method of claim 4 wherein maintaining the virtual clock comprises initializing the virtual clock to an initial value when the thread is created.

7. (Original) The method of claim 6 wherein maintaining the virtual clock comprises initializing the virtual clock to zero when the thread is created.

8. (Previously Presented) The method of claim 4 wherein maintaining the set of thread segments that are ordered before the current thread segment of the thread comprises

maintaining a set of ordered pairs, wherein one member of a pair is a thread identifier, and the other member of the pair is a virtual clock value.

9. (Canceled)

10. (Currently Amended) The method of claim [[9]]4, wherein incrementing the virtual clock associated with the ~~first~~ prior thread comprises incrementing the virtual clock associated with the ~~first~~ prior thread by one.

11. (Currently Amended) The method of claim 9 wherein initializing the virtual clock associated with the ~~second~~ new thread comprises initializing the virtual clock associated with the ~~second~~ new thread to zero.

12. (Previously Presented) The method of claim 8, further comprising:

if a first thread joins a forked thread, computing a set of thread segments that are ordered before the current thread segment of the first thread as the union of:

(a) the set of thread segments that are ordered before the current thread segment of the first thread, (b) the set containing the thread segments that are ordered before the current thread segment of the forked thread but which do not belong to the first thread, and (c) a singleton set containing the current thread segment of the forked thread.

13. (Original) The method of claim 12 wherein the set containing the thread segments that are ordered before the current thread segment of the forked thread but which do not belong to the first thread comprises a set containing the ordered pairs in the set of thread segments that are ordered before the current thread segment of the forked thread, such that the thread identifiers in the ordered pairs do not represent the first thread.

14. (Original) The method of claim 8, further comprising, if a thread accesses a shared memory location:

updating the set of concurrent thread segments that access the location by forming a set comprising the union of (a) a set containing the current thread segment of the thread, and

(b) a set containing the thread segments in the set of concurrent thread segments that continue to access the location; and

if the updated set of concurrent thread segments contains at most one element, then updating the set of locks associated with the location to the set of locks associated with the thread, and otherwise:

(i) updating the set of locks associated with the location to a set comprising the intersection of (a) the set of locks associated with the location and (b) the set of locks associated with the thread, and

(ii) if the set of locks associated with the location is empty, reporting a warning of a potential race condition.

15. (Original) The method of claim 14 wherein the set containing the thread segments in the set of concurrent thread segments that continue to access the location is formed by computing a subset of the set of concurrent thread segments, wherein the subset contains each thread segment a that satisfies the following predicate:

for every thread segment b in the set of thread segments ordered before a, at least one of the following is true: (i) the thread identifier of a is not equal to the thread identifier of b and (ii) the virtual clock value of a is greater than the virtual clock value of b.

16. (Currently Amended) A dynamic race detection system, comprising:

a hardware means for modifying and implementing a compiler of a runtime system that inserts calls to a race detector in compiled code, wherein the calls to the race detector are dynamically inserted by a JIT ("Just-In-Time") compiler; and

a hardware means for implementing a memory allocator of the runtime system that adds to shared memory objects instrumentation information required by the race detector, wherein the instrumentation information indicates instructions for invoking the race detector.

17. (Original) The system of claim 16 wherein the compiler is a modification of another compiler.

18. (Original) The system of claim 16 wherein the memory allocator is an alteration of another memory allocator.

19. (Currently Amended) A computer-implemented method for dynamic race detection, comprising:

a hardware means for, by way of modifying a compiler of a runtime system, inserting calls to a race detector in compiled code, wherein the calls to the race detector are dynamically inserted by a JIT ("Just-In-Time") compiler; and

a hardware means for, by way of a memory allocator of the runtime system, adding instrumentation information required by the race detector to shared memory objects, wherein the instrumentation information indicates instructions for invoking the race detector.

20. Canceled.

21. (Currently Amended) The method of claim [[20]] 19 wherein adding the instrumentation information required by the race detector is by way of changing the memory allocator of the runtime system.

22. (Currently Amended) A system for dynamically detecting potential race conditions in a program having a plurality of threads and one or more shared memory locations, the system comprising:

with respect to each shared memory location, (i) a mechanism for maintaining a set of concurrent thread segments that access said shared memory location, and (ii) a mechanism for maintaining a first set of locks associated with said shared memory location;

with respect to each thread, (i) a mechanism for maintaining a set of thread segments that are ordered before a current thread segment of a thread, (ii) a mechanism for maintaining a second set of locks that are acquired and released by the thread, and maintaining a virtual clock associated with each thread; and

a mechanism for modifying a compiler to insert calls to a race detector in compiled code, wherein the calls to the race detector are dynamically inserted by a JIT ("Just-In-

Time") compiler wherein the race detector reports a warning when a potential race condition is detected.